UBI 文件系统使用指南

文档版本　04

发布日期　2018-11-30

商标声明

 、HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址：　深圳市龙岗区坂田华为总部办公楼　邮编：518129

网址：　http://www.hisilicon.com/cn/

客户服务邮箱：　support@hisilicon.com

前 言

概述

linux-2.6.27 后，内核加入了一种新型的 FLASH 文件系统 UBI (Unsorted Block Images)。主要针对 FLASH 的特有属性，通过软件的方式来实现日志管理、坏块管理、损益均衡等技术。

本文主要介绍如何在内核中配置使用 UBI 文件系统以及制作对应的 UBI 文件系统根文件系统镜像。同时还介绍了如何转换镜像格式以便于在 u-boot 上进行烧录。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3519	V100
Hi3519	V101
Hi3516C	V300
Hi3516D	V200
Hi3516E	V100
Hi3516E	V200
Hi3516E	V300
Hi3518E	V300
Hi3559	V100
Hi3556	V100
Hi3516A	V200
Hi3536C	V100
Hi3559A	V100ES
Hi3559A	V100
Hi3559C	V100

产品名称	产品版本
Hi3536D	V100
Hi3531D	V100
Hi3521D	V100
Hi3520D	V400
Hi3521A	V100
Hi3531A	V100
Hi3518E	V200
Hi3518E	V201
Hi3516C	V200
Hi3519A	V100
Hi3556A	V100
Hi3516D	V300
Hi3516A	V300
Hi3516C	V500
Hi3559	V200
Hi3556	V200
Hi3516E	V200
Hi3516E	V300
Hi3518E	V300
Hi3516D	V200

读者对象

本文档（本指南）主要适用于技术支持工程师。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 04 (2018-11-30)

新增 Hi3516E V200/Hi3516E V300/Hi3518E V300。

文档版本 03 (2018-05-20)

2.2、2.4 和 2.5 小节涉及修改。

文档版本 02 (2018-01-16)

新增 Hi3559AV100/Hi3559CV100。

文档版本 01 (2017-12-20)

新增 Hi3520DV400/Hi3521DV100/Hi3531DV100。

文档版本 00B08(2017-11-20)

2.2 小节和 2.4 小节，涉及更新。

新增 2.5 小节。

文档版本 00B07(2017-08-15)

添加 Hi3536DV100 的相关内容。

文档版本 00B06 (2017-05-27)

添加 Hi3559AV100ES 的相关内容。

文档版本 00B05 (2017-04-10)

添加 Hi3536CV100 的相关内容。

文档版本 00B04 (2017-03-27)

2.2 和 3.2 小节涉及修改

文档版本 00B03 (2017-02-25)

添加 Hi3556V100 的相关内容。

2.1 小节涉及修改

文档版本 00B02 (2016-07-30)

添加 Hi3516CV300/Hi3559V100 的相关内容。

文档版本 00B01 (2015-11-04)

第 1 次临时版本发布。

目 录

1 内核中使能 UBI

芯片内核版本明细，请参考表 1-1 所示。

表1-1 芯片内核版本

芯片	内核版本
Hi3519V100/Hi3519V101/Hi3516CV300/Hi3516EV100/Hi3559V100/Hi3556V100/Hi3516AV200/Hi3559AV100ES/Hi3521DV100/ Hi3531DV100/Hi3520DV400/Hi3536CV100	linux3.18.y
Hi3536DV100/Hi3521AV100/ Hi3531AV100/Hi3518EV200/ Hi3518EV201/ Hi3516CV200/Hi3559AV100/Hi3559CV100/Hi3519AV100/Hi3556AV100/Hi3516CV500/Hi3516DV300/ Hi3516AV300/Hi3559V200/Hi3556V200/Hi3516EV200/Hi3516EV300/Hi3518EV300/Hi3516DV200	linux4.9.y

注意

本文档中以单板上的内核版本 linux3.18.y 为例，使用 UBIFS 文件系统，可以按以下配置。

1.2 内核配置 UBI 选项

步骤 1 使能 UBI 设备驱动

```
Device Drivers  --->
<*> Memory Technology Device (MTD) support  --->
   <*> Enable UBI - Unsorted block images  --->
```



```
-*- Enable UBI - Unsorted block images
(4096) UBI wear-leveling threshold
(20)  Maximum expected bad eraseblock count per 1024 eraseblocks
[ ]   UBI Fastmap (Experimental feature)
<*>   MTD devices emulation driver (gluebi)
[ ]   Read-only block devices on top of UBI volumes
```

说明

必须先使能 UBI 设备驱动，才能找到 UBIFS 文件系统选项。

步骤 2 使能 UBIFS 文件系统

```
File systems  --->
    -*- Miscellaneous filesystems  --->
        <*>   UBIFS file system support
```

```
<*>   UBIFS file system support
[ ]     Advanced compression options (NEW)
```

说明

所有配置按以上图中所示，其它 UBI/UBIFS 配置选项使用系统默认值，不要随意选择配置，如果选择不慎，UBI 文件系统可能无法正常工作。

----结束

1.3 UBI 设备驱动配置选项说明

- UBI wear-leveling threshold

  UBI 系统记录每个擦除块发生擦除操作的次数。此选项表示所有擦除操作次数中，最小值和最大值之间允许的最大间隔。此值默认为 4096，对于寿命比较短的 MLC 器件，此值应该配置相对小一点，比如 256。

- MTD devices emulation driver (gluebi)

  模拟 MTD 驱动，选择此选项，当创建一个卷时，UBI 将同时模拟一个 MTD 设备。这个功能提供了一个接口,供其它文件系统使用 UBI。

2 UBIFS 应用样例

2.1 mount 一个空 UBIFS 文件系统

单板当前有 4 个分区，分区的情况如下图。

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 01000000 00020000 "hinand"
mtd1: 00400000 00020000 "kernel"
mtd2: 02000000 00020000 "rootfs"
mtd3: 03200000 00020000 "ubi"
```

通过以下几步，就可以把 mtd3 分区映射成 ubi 卷，做为 ubi 分区使用。

步骤 1 格式化 ubi 分区

使用以下命令格式化 ubi 分区。

```
# ubiformat /dev/mtd3
```

说明

- OSDRV 编译完成后，生成的 UBI 工具放在 osdrv/pub/bin/board_xxx/目录下，board_xxx 路径的命名与编译时选择的工具链以及产品相关。
- 需将 UBI 工具下载到单板，通过命令“chmod +x ‘ubi 工具’”加可执行权限。
- 不推荐擦除(如:用命令 flash_eraseall)分区，擦除分区后，可以正常 mount 到 ubifs。 但是擦除分区操作，将使 UBI 系统丢失记录的每个擦除块的擦除次数。
- MTD 分区大小至少要 44 个 block。

步骤 2 绑定 UBI 到 MTD 分区

绑定 UBI 到 mtd3 分区，使用以下命令。


```
# ubiattach /dev/ubi_ctrl -m 3
```

参数”-m 3”表示使用 mtd3 分区。只有绑定了 ubi 到 mtd 分区以后，才能在 /dev/ 下找到 ubi 设备“ubi0”，如果曾经创建过 ubi 卷，那么绑定以后才能在 /dev/ 下找到并且访问 ubi 卷 “ubi0_0”。

命令执行成功，显示信息如下图。

```
# ubiattach /dev/ubi_ctrl -m 3
UBI: attaching mtd3 to ubi0
UBI: scanning is finished
UBI: attached mtd3 (name "UBIFS01", size 50 MiB) to ubi0
UBI: PEB size: 131072 bytes (128 KiB), LEB size: 126976 bytes
UBI: min./max. I/O unit sizes: 2048/2048, sub-page size 2048
UBI: VID header offset: 2048 (aligned 2048), data offset: 4096
UBI: good PEBs: 400, bad PEBs: 0, corrupted PEBs: 0
UBI: user volume: 0, internal volumes: 1, max. volumes count: 128
UBI: max/mean erase counter: 1/1, WL threshold: 4096, image
sequence number: 728242785
UBI: available PEBs: 376, total reserved PEBs: 24, PEBs reserved
for bad PEB handling: 20
UBI: background thread "ubi_bgt0d" started, PID 101
UBI: attaching mtd3 to ubi0
UBI: scanning is finished
UBI: attached mtd3 (name "UBIFS01", size 50 MiB) to ubi0
UBI: PEB size: 131072 bytes (128 KiB), LEB size: 126976 bytes
UBI: min./max. I/O unit sizes: 2048/2048, sub-page size 2048
UBI: VID header offset: 2048 (aligned 2048), data offset: 4096
UBI: good PEBs: 400, bad PEBs: 0, corrupted PEBs: 0
UBI: user volume: 0, internal volumes: 1, max. volumes count: 128
UBI: max/mean erase counter: 1/1, WL threshold: 4096, image
sequence number: 728242785
UBI: available PEBs: 376, total reserved PEBs: 24, PEBs reserved
for bad PEB handling: 20

UBI: background thread "ubi_bgt0d" started, PID 101
```

最后一行打印“ubi_bgt0d”表示成功创建设备 ubi0，查看所有设备“ls /dev/ubi*”，将发现多一个设备“/dev/ubi0”。

步骤 3 创建 UBI 卷

UBI 卷可以理解为 UBI 设备的分区。创建 ubi 卷命令如下：

```
# ubimkvol /dev/ubi0 -N ubifs -s  SIZE
```

参数“/dev/ubi0”是上一步骤创建的 ubi 设备。

参数“-N ubifs”表示创建的卷名为“ubifs”。

参数“-s SIZE”表示创建的分区大小。

说明

SIZE 值应小于“/dev/ubi0”设备能提供的空间大小。

可以使用命令“ubinfo”查看当前可使用的 LEBs 大小。如下图红色标记行所示，当 UBI 设备提供的空间为 50 MiB 时，可使用的空间大小为 45.5MiB。所以，应该保证所创建的卷的 SIZE 值小于可使用的 LEBs 空间大小。

```
# ubinfo /dev/ubi0
ubi0
Volumes count:                           0
Logical eraseblock size:                 126976 bytes, 124.0 KiB
Total amount of logical eraseblocks:     400 (50790400 bytes, 48.4 MiB)
Amount of available logical eraseblocks: 376 (47742976 bytes, 45.5
MiB)
Maximum count of volumes                 128
Count of bad physical eraseblocks:       0
Count of reserved physical eraseblocks:  20
Current maximum erase counter value:     1
Minimum input/output unit size:          2048 bytes
Character device major/minor:            253:0
```

查看所有设备“ls /dev/ubi*”，将发现多一个设备“/dev/ubi0_0”。

说明

卷只用创建一次，创建后，卷信息将被记录在 UBI 设备上，下一次启动，不用再次创建卷。删除卷，用命令“ubirmvol”. 如果使用此命令删除卷，卷上所有数据，也将被删除。

步骤 4 挂载空 UBIFS 文件系统

此时就可以将创建的卷挂载到指定的目录上去了，命令如下：

```
# mount -t ubifs /dev/ubi0_0 /mnt/
```

或者

```
# mount -t ubifs ubi0:ubifs /mnt/
```


参数“/dev/ubi0_0”表示 mount 到卷 “ubi0_0”，也可以使用参数“ubi0:ubifs”。某些版本的内核，不支持“/dev/ubi0_0”形式的参数，只能使用“ubi0:ubifs”形式的参数. “ubi0:ubifs”中的“ubifs”表示卷的名称, 在创建 ubi 卷时设置。

Mount 成功，将显示如下信息：

```
# mount -t ubifs /dev/ubi0_0 /mnt/
UBIFS: default file-system created
UBIFS: background thread "ubifs_bgt0_0" started, PID 107
UBIFS: mounted UBI device 0, volume 0, name "ubifs"
UBIFS: LEB size: 126976 bytes (124 KiB), min./max. I/O unit sizes:
2048 bytes/2048 bytes
UBIFS: FS size: 46473216 bytes (44 MiB, 366 LEBs), journal size
2285568 bytes (2 MiB, 18 LEBs)
UBIFS: reserved for root: 2195044 bytes (2143 KiB)
UBIFS: media format: w4/r0 (latest is w4/r0), UUID 80EC88B4-1AF1-4193-
AC8F-5506B1A21742, small LPT model
```

查看分区信息，将显示如下内容：

```
# df -h
Filesystem              Size      Used Available Use% Mounted on
/dev/root              32.0M     14.8M     17.2M  46% /
devtmpfs               28.9M      4.0K     28.9M   0% /dev
/dev/ubi0_0            40.3M     20.0K     38.2M   0% /mnt
```

UBIFS 文件系统显示的分区大小、剩余空间并不准确。因为 UBIFS 文件保存的是文件压缩后的内容，压缩比率与文件内容相关。可能剩余空间显示只有 2M, 但是可以将一个 4M 的文件完整保存。

----结束

2.2 制作 UBIFS 根文件系统 UBI 镜像

制作 ubifs 文件系统镜像，需要使用 mtd-utils 工具，命令如下：

```
$./mkfs.ubifs -F -d rootfs_uclibc -m 2KiB -o rootfs.ubiimg -e 126976 -
c 256 -v
```

参数“-F”使能“white-space-fixup”，如果是通过 u-boot 烧写需要使能此功能。

参数“-d rootfs_uclibc”表示将要被制作为 UBIFS 镜像的根目录为“rootfs_uclibc”，这个参数也可以写为 “-r rootfs_uclibc”。

参数“-m 2KiB”表示最小读写单元是 2KiB，这个参数也可以写为“-m 2048”。这里使用的 NAND 芯片页大小为 2KiB。最小读写单元是指 FLASH 器件一次读写操作，最小操作的字节数，对 NAND 器件，是页大小，如 2K/4K/8K；对于 NOR 器件，是 1 个字节。

参数“-o rootfs.ubiimg”表示制作出来的镜像名称为“rootfs.ubiimg”。

参数 “-e 126976”表示逻辑擦除块大小。

最小读写单元和逻辑擦除块大小可以通过读 MTD 和 UBI 系统信息获得，也可以通过计算获得。读 MTD 信息命令以及显示内容如下：

```
# mtdinfo /dev/mtd3
mtd3
Name:                           ubi
Type:                           nand
Eraseblock size:                131072 bytes, 128.0 KiB
Amount of eraseblocks:          400 (52428800 bytes, 50.0 MiB)
Minimum input/output unit size: 2048 bytes
Sub-page size:                  2048 bytes
OOB size:                       60 bytes
Character device major/minor:   90:6
Bad blocks are allowed:         true
Device is writable:             true
```

读 UBI 信息命令（此命令需要先绑定 UBI 见 2.1.3）以及显示内容如下：

```
# ubinfo /dev/ubi0
ubi0
Volumes count:                       1
Logical eraseblock size:             126976 bytes, 124.0 KiB
Total amount of logical eraseblocks:    400 (50790400 bytes, 48.4
MiB)
Amount of available logical eraseblocks: 0 (0 bytes)
Maximum count of volumes             128
Count of bad physical eraseblocks:      0
Count of reserved physical eraseblocks:  20
Current maximum erase counter value:    2
Minimum input/output unit size:         2048 bytes
Character device major/minor:           253:0
Present volumes:                     0
```


参数“-c 256”表示此文件系统最多使用“256”个逻辑擦除块。计算“256 * LEB”得到此文件系统的最大可使用空间。

参数“-v”显示制作 UBIFS 过程中的详细信息。

以上标记为红色的，表示此芯片的逻辑擦除块大小。

逻辑擦除块大小也可以通过计算得到，计算方法如下表：

FLASH 种类	逻辑擦除块(LEB)大小
NOR	LEB = blocksize – 128
NAND 无子页	LEB = blocksize – pagesize * 2
NAND 有子页	LEB = blocksize – pagesize * 1
说明: Blocksize : flash 物理擦除块大小; Pagesize: flash 读写页大小;	

下图为制作成功后, “mkfs.ubifs”的详细打印。


```
$ ./mkfs.ubifs -d rootfs_uclibc -m 2KiB -o rootfs.ubiimg -e 126976 -c
256 -v
mkfs.ubifs
        root:        rootfs_uclibc/
        min_io_size:  2048
        leb_size:     126976
        max_leb_cnt:  256
        output:       rootfs.ubiimg
        jrn_size:     3936256
        reserved:     0
        compr:        lzo
        keyhash:      r5
        fanout:       8
        orph_lebs:    1
        space_fixup:  0
        super lebs:   1
        master lebs:  2
        log_lebs:     4
        lpt_lebs:     2
        orph_lebs:    1
        main_lebs:    45
        gc lebs:      1
        index lebs:   1
        leb_cnt:      55
```

这里需要注意，制作成功的 UBIFS 根文件系统镜像为 UBI 镜像，可以在内核下对空 UBIFS 文件系统进行升级（update）操作，详见 2.3 空 UBIFS 文件系统升级为根文件系统。

该镜像不能直接烧录到 MTD 分区上使用，但是可以通过格式转换,转换成能直接烧录 MTD 分区上的格式。详见 2.4 UBI 镜像的转换格式和烧录。

说明

做 UBI 镜像需要专用工具和内核版本必须搭配使用，UBI 工具版本号和单板内核版本如果不配套,制作的镜像无法在单板上 mount。

2.3 空 UBIFS 文件系统升级为根文件系统

在内核(区别 u-boot)下建立好 UBI 卷后，可以使用应用程序，直接对卷进行升级。升级步骤如下：

步骤 1 创建建立 UBI 卷

详见 2.1.3 创建 UBI 卷

步骤 2 制作 UBIFS 根文件系统 UBI 镜像

详见 2.2 制作 UBIFS 根文件系统 UBI 镜像。

步骤 3 tftp 下载根文件系统 UBI 镜像到内核

```
# tftp -g -r rootfs.ubiimg 10.67.209.140
```

步骤 4 在内核下升级 UBIFS 文件系统

使用以下命令：

```
# ubiupdatevol /dev/ubi0_0  rootfs.ubiimg
```

参数“/dev/ubi0_0”表示需要升级的卷，这个卷需要预先创建，升级前，卷上的内容可以不用擦除。

清除卷内容，使用命令“ubiupdatevol /dev/ubi0_0 –t”。

步骤 5 设置 u-boot 启动参数

UBIFS 下 u-boot 的启动参数 bootargs 配置形式如下图：

```
setenv bootargs 'mem=128M console=ttyAMA0,115200 ubi.mtd=3
root=ubi0:ubifs rootfstype=ubifs rw
mtdparts=hinand:1M(boot),4M(kernel),32M(yaffs2),50M(ubi),-
(reserve)'
```

参数“ubi.mtd=3”表示 UBI 绑定到“/dev/mtd3”分区。

参数“root=ubi0:ubifs”中“ubi0”表示使用 UBI 绑定后的 UBI 分区，其中“ubifs”为创建 UBI 卷时定义的卷名。某些内核版本不识别“root=/dev/ ubi0_0”形式的参数。

参数“rootfstype=ubifs”表示使用 ubifs 文件。

----结束

2.4 UBI 镜像的转换格式和烧录

步骤 1 制作 UBIFS 根文件系统 UBI 镜像

详见 2.2 制作 UBIFS 根文件系统 UBI 镜像。

步骤 2 制作 UBI 镜像转换配置文件

UBI 镜像转换格式时，需要一个配置文件 ubi.cfg 作为第（3）步的输入。内容如下图所示：

```
[ubifs-volumn]
mode=ubi
image=./rootfs_hi35xx_2k_128k_32M.ubiimg
vol_id=0
vol_type=dynamic
vol_alignment=1
vol_name=ubifs
vol_flags=autoresize
```

参数“mode=ubi”是强制参数，当前不能输入别的值，保留为以后扩展功能；

参数“image=./rootfs*.ubiimg”表示卷对应的 UBIFS 文件系统镜像文件名称，此文件即 2.2 制作 UBIFS 根文件系统 UBI 镜像制作的镜像文件。

参数“vol_id=0”表示卷的 ID 号, UBI 镜像可能包含多个卷，这个用来区别不同的卷。

参数“vol_type=dynamic”表示当前卷类型是可读写的。如果此文件为只读,对应的参数应该为“vol_type=static”；

参数“vol_name=ubifs”表示卷的名称, UBIFS 做根文件系统时，将用到卷名称。

参数“vol_flags=autoresize”表示卷大小是可以动态扩展。

步骤 3 转换 UBI 镜像格式

使用以下命令：

```
$ ./ubinize -o rootfs.ubifs -m 2KiB -p 128KiB ubi.cfg -v
```

参数“-o rootfs.ubifs”表示输出的 UBI 镜像转换后的名称为“rootfs.ubifs”，输入的 UBI 镜像文件名由 ubi.cfg 配置文件输入。

参数“-m 2KiB”表示最小读写单元是“2KiB”。

参数“-p 128KiB”表示 flash 的擦除块大小。注意这是物理擦除大小，不是逻辑擦除块大小。

参数“ubi.cfg”是一个配置文件，第（2）步已经详细讲解过此文件。

参数“-v”显示制件过程的详细信息。

下图为制作成功后的详细打印。


```
$ ./ubinize -o rootfs.ubifs -m 2KiB -p 128KiB ubi.cfg -v
ubinize: LEB size:               126976
ubinize: PEB size:               131072
ubinize: min. I/O size:           2048
ubinize: sub-page size:           2048
ubinize: VID offset:             2048
ubinize: data offset:             4096
ubinize: UBI image sequence number: 2067745235
ubinize: loaded the ini-file "ubi.cfg"
ubinize: count of sections: 1
ubinize: parsing section "ubifs-volumn"
ubinize: mode=ubi, keep parsing
ubinize: volume type: dynamic
ubinize: volume ID: 0
ubinize: volume size was not specified in section "ubifs-volumn",
assume minimum to fit image
"./rootfs_hi35xx_2k_128k_32M.ubiimg"6983680 bytes (6.7 MiB)
ubinize: volume name: ubifs
ubinize: volume alignment: 1
ubinize: autoresize flags found
ubinize: adding volume 0
ubinize: writing volume 0
ubinize: image file: ./rootfs_xxx_2k_128k_32M.ubiimg
ubinize: writing layout volume
ubinize: done
```

步骤 4 U-BOOT 下 ubi 镜像转换文件的烧写

U-BOOT 下，烧写 UBI 镜像转换文件和烧写内核的方法一样。 命令如下图所示：

```
# nand erase [offset] [len]
# mw.b [ddr_addr] 0xff [len]
# tftp [ddr_addr] rootfs.ubifs
# nand write [ddr_addr] [flash_start_addr] [ubi_len]
```

说明

- offset 即进行 flash 操作的开始地址，例 nand erase 0x800000 0x720000，即 flash 从 8MB 开始擦除，擦除 7296KB 长度。
- len 是 UBIFS 根文件系统分区大小长度。
- ddr_addr 即内存地址，要选用可使用的 ddr 地址进行操作，否则可能会造成系统挂死。具体项目 ddr 地址请参考《Hi35xx U-boot 移植应用开发指南》中烧写方法。

----结束

2.5 使用 mkubiimg.sh 脚本一键式制作 UBI 镜像

针对 2.1-2.4 章节繁琐的 UBI 镜像和 UBIFS 根文件系统镜像制作步骤，在 osdrv/tools/pc/ubi_sh/目录下提供了 mkubiimg.sh 脚本，专门用于制作无子页 Nand Flash 使用的 UBI 镜像和 UBIFS 根文件系统镜像。

使用以下命令：

```
$./mkubiimg.sh  Chip  Pagesize  Blocksize  Dir  Size  Tool_Path  Res
```

参数说明：

Chip.	芯片名称：如hi35xx
Pagesize	NAND page size. 2k/4k/8k.
Blocksize	NAND block size. 128k/256k/1M
Dir	将要被制作为UBIFS镜像的根目录
Size	UBIFS根文件系统分区的大小.
Tool_Path	制作UBI镜像所需mkfs.ubifs和ubinize工具的存放目录，一般放于osdrv/pub/bin/pc/目录
Res	是否保留UBI镜像和ubicfg配置文件 (1:Yes 0:No(默认))

说明

举个例子，执行完命令：./mkubiimg.sh hi35xx 2k 128k osdrv/pub/rootfs 50M osdrv/pub/bin/pc 1 之后会生成三个文件：

- rootfs_hi35xx_2k_128k_50M.ubiimg：该镜像不能直接烧录到 MTD 分区，但是可以使用 ubiupdate 命令在内核下对空 UBIFS 文件系统进行升级操作，详见 2.3 “空 UBIFS 文件系统升级为根文件系统”。
- rootfs_hi35xx_2k_128k_50M.ubicfg：UBI 镜像转换格式时，需要一个配置文件。这个配置文件主要指明根文件系统的卷名，卷类型和扩展属性等内容。
- rootfs_hi35xx_2k_128k_50M.ubifs：该镜像可以直接烧录到 Nand Flash，是.ubiimg 转换格式的最终镜像。

3 附录

3.1 UBI 和 MTD 相关的接口和命令

(1) cat /proc/mtd	可以看到当前系统的各个 mtd 情况。
(2) mtdinfo /dev/mtd3	查看 MTD 分区信息。
(3) ubinfo –a	显示当前所有 UBI 分区信息
(4) ls /dev/ubi*	查看 UBI 设备节点和 UBI 卷

3.2 UBI 常见问题

- 空 FLASH 运行 UBI 前，必须要先用 erase 命令擦除么?

  没有必要,UBI 文件系统会记录每个块的使用次数,如果 ERASE,将会把这些使用次数也擦除掉。这将破坏 UBI 的读写均衡特性。使用 ubiformat 程序，能擦除 flash，并且保存每个块的读写次数。

- 为什么第一次能 mount 上 ubifs，重启以后会挂载失败?

  在内核版本在 3.0 以上，需要在制作 UBIFS 根文件系统 UBI 镜像的时候，增加“-F”参数，这个参数会设置修正空白区域的标志，在第一次 mount 的时候对空白区域进行修正，以便后续能正常使用。

- 为什么对 UBI 分区进行裸写操作后，下次重启后直接执行 attach 命令会出现 ECC 报错？

  UBIFS 对分区数据格式有严格要求，若对分区进行了裸写操作会破坏数据的格式。所以下次如果想要执行 attach 命令，必须先执行 ubiformat（或 flash_erase）命令，格式化分区，裸写后进行 UBI 操作必须先擦除。

